Forecasting GPU Performance for Deep Learning Training and Inference

基本信息

Conference：ASPLOS' 25

Authors：

- Seonho Lee seonho.lee@gatech.edu Georgia Institute of Technology Atlanta, GA, USA
- Amar Phanishayee aphanishayee@meta.com Meta Seattle, WA, USA
- Divya Mahajan divya.mahajan@gatech.edu Georgia Institute of Technology Atlanta, GA, USA

Background

- GPU 在 AI 集群中的主导地位（并行计算 + 软件优化）
- 新模型与新硬件迭代的脱节（交付周期长、成本高）

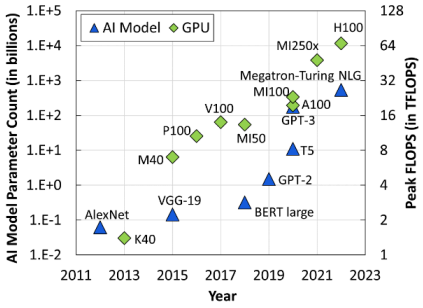


Figure 1. Growth of AI models and the compute and memory capacity of GPUs. [2, 7, 11, 22, 39, 50, 51, 54, 55]

- 性能预测需求（latency）
  - 新模型在现有 GPU 上的表现
  - 现有模型在新 GPU 上的表现
  - 新模型在新 GPU 上的表现

> 在本文中，深度神经网络（DNN）内核是一种张量算子，如卷积（CONV）、通用矩阵乘法（GEMM）、加法（Add）、Softmax 等，一旦启动，这些算子会在设备上完整执行。
>
> 对算子进行分类：
>
> - General Matrix
> - Multiplication (GEMM), fully-connected layer, and point wise
> - operators (ReLU, Tanh, Softmax, etc.).
>
> 假设：内核在设备上按顺序执行

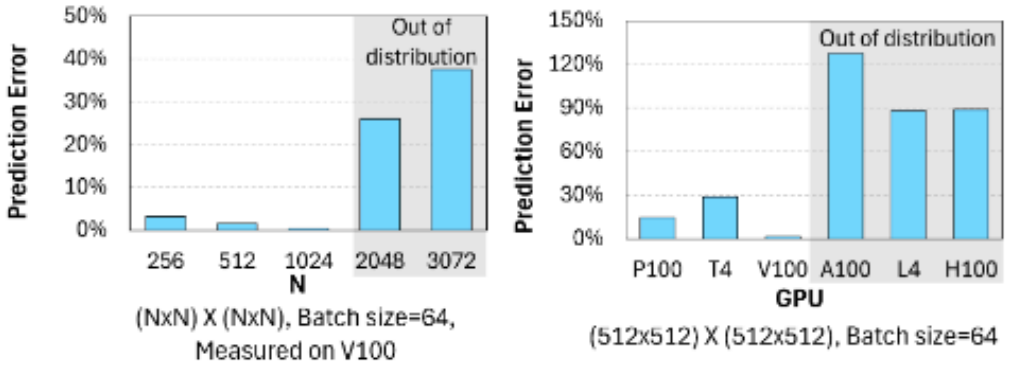
研究现状

局限性

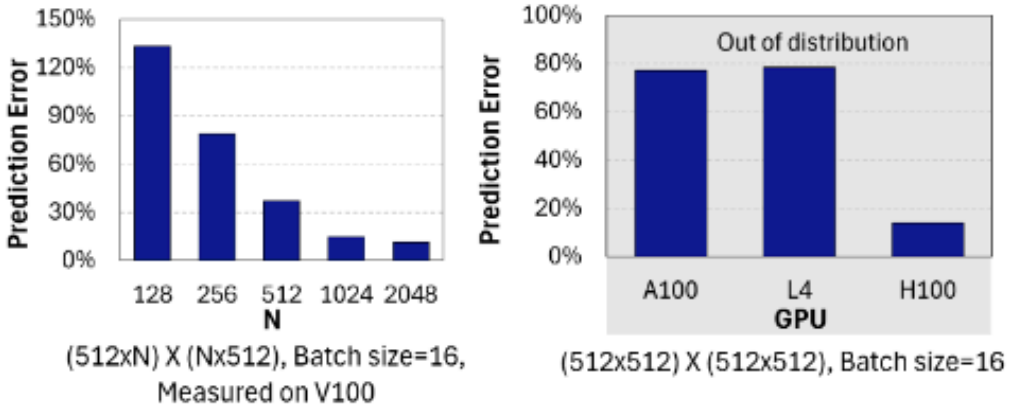
- **直接 ML 预测**：Habitat 直接训练 MLP 预测整体延迟，在泛化未见过模型（具有更大的矩阵维度） / 训练集以外的GPU 上误差高（图 2a）。
    - 对于 kernel-alike operator，在现有 GPU 上执行应用程序测量延迟，并根据目标 GPU 与现有 GPU 的硬件配置比例缩放延迟。
    - 对于 kernel-varying operator，训练多个多层感知机（MLP）模型来预测不同算子的延迟。MLP 的输入特征包括：内存大小、内存带宽、流多处理器核心数量、峰值浮点运算次数（FLOPS），以及内核输入输出维度；输出为实测延迟。

    > 算子的分类
    >
    > - kernel-varying operator（例如：GMM 批矩阵乘法）：由于硬件架构（SM、缓存大小、内存带宽）和优化策略（分块大小、线程分配）不同，同一算子（如矩阵乘法）在不同 GPU 架构（如 NVIDIA 的 A100 与 AMD 的 MI250）上通过不同的内核代码实现
    > - kernel-alike operator（例如：vector operator，**加法**、**ReLU**、**GELU** ）

- **线性回归假设**：假设延迟与 FLOPs 线性相关，构建 latency 和 计算量FLOPs 的线性回归模型；对于不同 GPU，构建 GPU 内存带宽与实际 FLOPS 性能之间的线性回归关系。但忽略了内存带宽瓶颈和硬件利用率波动（图 2b）。
    - 当矩阵乘法的维度较小时，GPU 利用率不足，延迟并不随 FLOPs 线性增长，导致线性回归假设失效。



(a) Multi-Layer Perceptron Approach (Habitat) [62]



(b) Linear Regression-based Approach (Li et al.) [26]

Figure 2. Prediction error of prior work on BMM operator, reported in percentage error. Out of distribution dimensions and GPUs are highlighted. For these results, we trained Habitat and Li et al. models only up to V100, excluding any A100s, H100, and L4.

原因

- **复杂度爆炸**：GPU 执行涉及 SM 调度、内存层次、库优化等多层交互，难以用单一模型捕捉。这种复杂性往往无法通过仅使用有效内存带宽、计算利用率和峰值浮点运算次数等高层特征的机器学习模型来捕捉。
- **跨代泛化不足**：新硬件的微架构变化（如 H100 的 Transformer 引擎）无法通过旧数据训练的模型预测。

挑战

开发一种仅使用抽象硬件架构特征即可精确估算 GPU 性能的解决方案。

核心挑战：深度学习模型在 GPU 上的性能与底层硬件细节及软件优化（如 CuDNN 和 Cutlass 等 DNN 内核库提供的优化）紧密相关。

- 与专用硬件加速器 [9,13,20,27,28,53] 不同，GPU 的微架构交互（如 L1/L2 缓存、DRAM、内存 I/O 配置和 warp 调度）与执行特性之间存在复杂的相互作用。因此，尽管已有大量研究能够精确估算特定领域加速器上的 DNN 内核性能，但这些方法并不适用于 GPU 性能预测。

> **硬件加速器**指的是专门为加速特定任务（如深度学习）而设计的专用硬件设备，例如：

> - **TPU（Tensor Processing Unit）**：Google 开发的专用 AI 芯片，针对矩阵运算和深度学习优化。
> - **NPU（Neural Processing Unit）**：华为等公司推出的神经网络专用处理器。
> - **ASIC（Application-Specific Integrated Circuit）**：针对特定算法（如卷积神经网络）定制的芯片。
>
> 这些加速器通常具有固定的架构和优化的数据流，与通用 GPU 相比，其微架构交互（如缓存、线程调度）更简单，因此更容易通过分析模型（如公式或线性回归）预测性能

Motivation

- 仅仅依靠机器学习技术来预测内核延迟是不够的——模型无法捕捉到由于软硬件优化而带来的 GPU 持续性能提升。

Table 1. Various ML models on predicting latency of BMM. Prediction errors are reported in percentage error.

Predictor Architecture	Number of layers	In-distribution Prediction Error (%)	Out-of-distribution Prediction Error (%)
MLP	8	28.0	70.9
	16	22.3	81.4
Transformer	3	22.3	126.1
	6	21.0	86.4

- 把内核的执行过程分解成多个较小工作tile，利用机器学习模型来预测每个 tile 对设备的利用率，基于 GPU 架构定义的性能边界来计算延迟。——通过分解使得使机器学习模型能够在更小且更简单的任务上进行训练。
- 基于内核在GPU上按顺序执行的假设，根据在GPU上执行的模型的数据流汇总每个内核的预测延迟，以确定每个 GPU 的执行延迟。

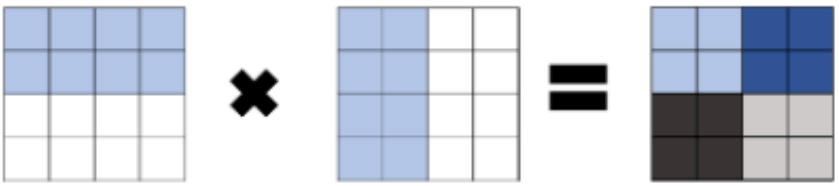
  Core idea：流行 GPU 库将深度学习内核分解为多个较小工作集tile并独立调度至 GPU 的 SM 上执行==>将端到端延迟预测分解为规则且更易管理的子问题，在 tile 粒度上进行延迟预测。

NeuSight Steps

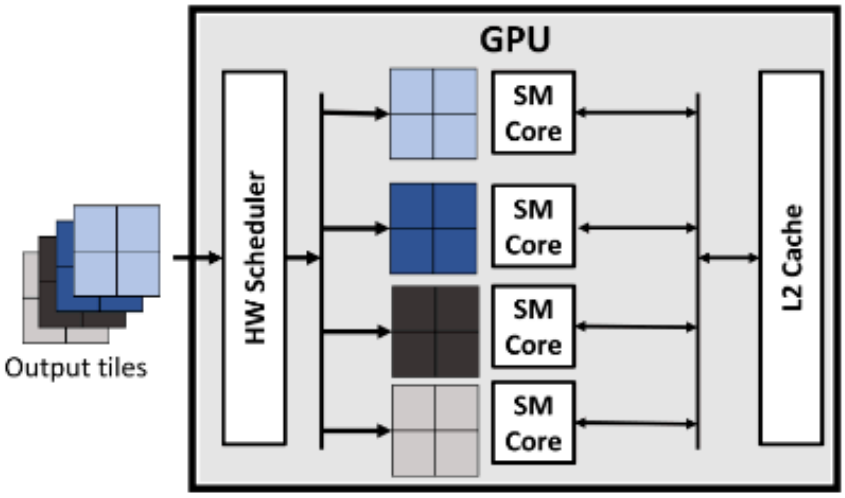
NeuSight 通过以下三个步骤预测深度学习模型在单 GPU 或多 GPU 服务器上的端到端延迟：

1. **内核级性能预测**：利用 GPU 的软件优化，例如分块（tiling）、跨流式多处理器（SMs）并行执行，以及硬件特性，例如内存容量、内存带宽、SM 数量、L2 缓存大小和峰值浮点运算次数（FLOPS）等，通过机器学习方法预测内核延迟；
2. **单 GPU 延迟聚合**：基于 DNN 的数据流图聚合各算子延迟，确定单 GPU 总延迟；
3. **分布式性能整合**：将单 GPU 执行性能与预测的网络延迟结合，计算服务器中多 GPU 协同执行 DNN 的性能。

NeuSight Forecasting



(a) Tiling of output matrix.



(b) Mapping tiles to multiple SMs.

Figure 3. Dataflow of a GEMM on a GPU. We assume multiplication between two 4x4 matrices and tile size of 2x2.

Kernel Execution on GPUs

Tiled execution

GEMM 步骤：

- 首先将输出矩阵分割为多个相同的分块（图 3a）。
- 每个分块对应输出矩阵的一个片段，从输入矩阵加载必要的操作数，并计算输出元素。这些分块被映射到 GPU 的每个流多处理器（SM）并发执行（图 3b）。
- 可并行执行的分块数量受限于 GPU 上 SM 的数量，整个 GPU 内核会像图 4 所示的那样，以多轮分块组的形式执行。

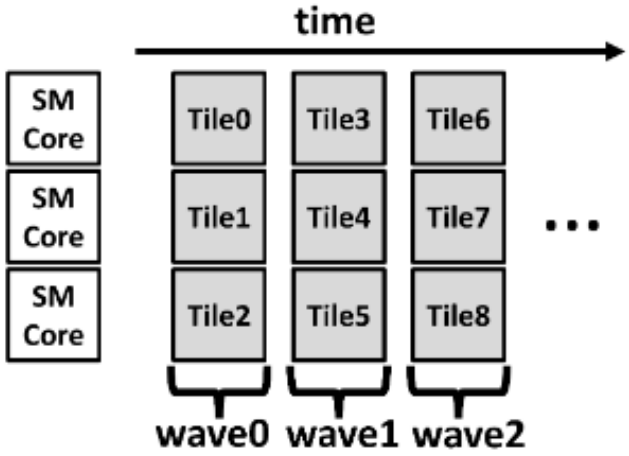
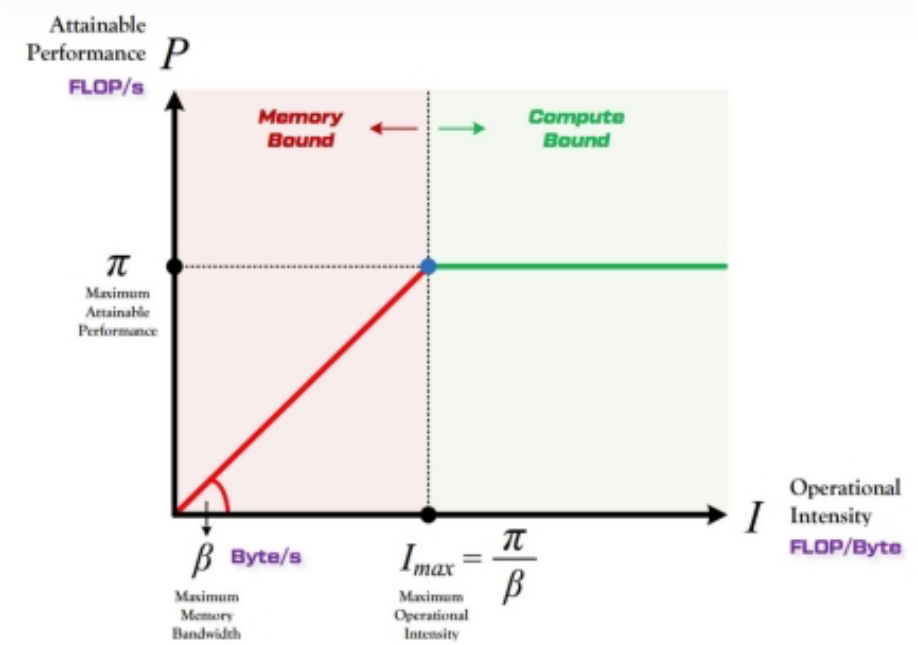


Figure 4. Each tile is distributed evenly across SMs and executed concurrently, in multiple number of waves. [33]

Fundamental performance laws of GPUs



根据 roofline 理论，roofline 计算带宽可以计算为：

$$K = \frac{flops_k}{mem_k};\ rooflin e_{BW} = min(K \times memBW_p, flops_p) \qquad (1)$$

min 的前一项表示完全受限于内存带宽时，理论上能够达到的计算能力，$rooflin e_{BW}$ 表示kernel 在 GPU 上能够达到的计算能力。

$memBW_p$表示硬件的峰值内存带宽；$flops_p$表示硬件的峰值计算能力

compute intensity K由浮点运算次数$flop_k$除以内存大小mem_k计算得出（**每字节内存操作对应的计算量**）

- 若 intensity 高（如矩阵乘法），性能瓶颈通常是计算资源（如 SM 的浮点运算能力）。
- 若 intensity 低（如内存带宽敏感的操作），性能瓶颈则是内存带宽。

Latency hiding in GPU

- 每个kernel并不总是能充分利用可用的计算或内存带宽

Table 2. Measured compute utilization of H100 GPU when executing (512x64) x (64x512) matrix multiplication across various batch sizes.

Batch Size	32	64	128	256	512
Peak FLOPS Utilization	53.2%	70.7%	69.4%	72.3%	86.0%

- GPU 性能在很大程度上依赖于kernel中的并行度水平，这与每个流多处理器（SM）中可独立调度的线程数量成正比

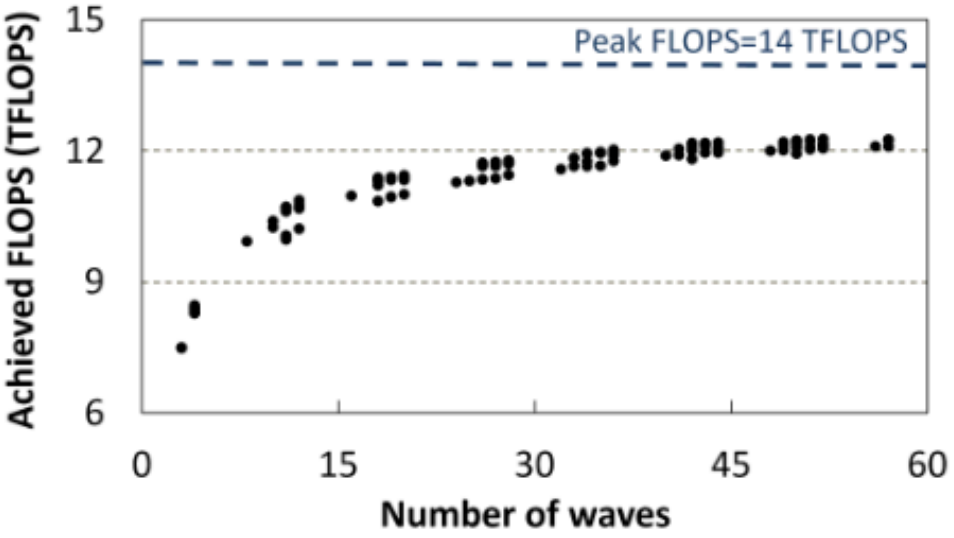


Figure 5. Performance of (256 × 256) × (256 × 256) matrix multiplication with varying waves on V100; sweeping the number of waves by changing the batch size from 1 to 300.

随着波次数量增加，每个 SM 被分配更多波次，导致每个 SM 中的线程数量增加，这使得每个线程的停顿延迟得以有效隐藏，从而提升性能。

Kernel-wise Prediction

根据分块执行以及 GPU 的基本性能定律对每个 kernel 的 latency 进行预测

Tile-granularity prediction

将 kennel 分解成相同的分块tiles，对分块进行预测，通过聚合这些预测结果来确定整个算子的延迟，而不是直接预测整个 kernel 的 latency。根据图4，kernel 执行的 latency 通常与波次数呈线性关系。

$$num_{tiles} = \Pi_{i=1}^{N} \lceil \frac{x_i}{t_i} \rceil \quad (2)$$

$$num_{waves} = \lceil \frac{num_{tiles}}{num_{sm}} \rceil \quad (3)$$

$$PerOpLatency = PerTileLatency \times num_{waves} \quad (4)$$

> **waves**
>
> 在 V100 执行 (256×256)×(256×256) 矩阵乘法时：
>
> - **分解为 tile**：生成 64×64=4096 个 tile（假设 tile 尺寸 32×32）。
> - **波次计算**：V100 有 80 个 SM，波次数量为⌈4096/80⌉=52波。一波有 80 个 tiles（也就是 80 个线程块，80 个 SM 并发执行这一波）
> - **执行方式**：每个 SM 处理 52 波中的 1 波，每波包含多个线程块，并发执行。实质上是每个 SM 都分配了 52 波，执行每波中的一个 tile

$PerTileLatency$（每个分块的 latency）依赖于 SM 的计算利用率，并受限于 roofline（**Fundamental performance laws of GPUs**）

$$PerTileLatency = \frac{flops_{tile}}{achieved_{BW}} \quad (5)$$

$$achieved_{BW} = roofline_{BW} \times utilization \quad (6)$$

Machine Learning Model to Predict Utilization

从图 5 可以看出 waves 增多，GPU 的吞吐量趋近于最大可实现值。原因是当工作负载的线程数增加时（如波次数量增多），GPU 通过多线程调度隐藏内存延迟，从而提升吞吐量。

因此通过公式 7 对利用率进行建模：

$$utilization = alpha - \frac{beta}{num_{waves}} \quad (7)$$

$$alpha, beta = \sigma(MLP(input\_features)) \quad (8)$$

训练了五个不同针对不同类型内核的 MLP预测器，分别针对批量矩阵乘法、全连接层、逐元素算子、softmax 和层归一化进行预测。

输入特征：显存大小、带宽、峰值浮点运算次数（FLOPS）以及 L2 缓存大小

输出结果：$beta$、$alpha$预测值

特征预处理：由于是在分块粒度上进行预测，分块会映射到单个 SM，因此将输入特征预处理为每个 SM 的资源量。

Table 3. List of input features for predicting the utilization.

Input Features	Unit
$\frac{FLOPsPerTile}{PeakFLOPSPerSM}$	$\frac{GFLOPS}{TFLOPS/s}$
$\frac{MemoryPerTile}{MemoryBWPerSM}$	$\frac{MB}{GB/s}$
$\frac{num_{waves} \times MemoryPerTile}{L2CacheSizePerSM}$	$\frac{MB}{KB}$
$\frac{num_{waves} \times MemoryPerTile}{MemorySizePerSM}$	$\frac{MB}{MB}$
$\frac{FLOPsPerTile \;/\; MemoryPerTile}{PeakFLOPS \;/\; MemoryBW}$	$\frac{GFLOPS \;/\; MB}{(TFLOPS/s) \;/\; (GB/s)}$

未知算子处理

对于未见过的算子，NeuSight 将其视为内存受限型任务，通过内存需求除以内存带宽估算延迟。

融合算子的预测

每个算子的浮点运算次数相加，忽略内核之间中间结果所需的内存大小

利用融合算子的浮点运算次数、内存大小，以及第一个算子的分块大小，通过第一个算子的MLP 预测器进行预测。

Forecasting for Distributed Execution

针对每种并行形式，NeuSight 会在深度神经网络（DNN）计算图中插入对应的网络算子：

- 流水线并行：在流水线阶段间传递激活值——根据流水线并行调度方案、microbatch 大小、GPU 数量，插入 send-receive operation；
- 数据并行：对梯度执行allreduce操作
- 张量模型并行：对激活值和梯度执行allreduce操作

对网络算子ring-based allreduce 和peer-to-peer send/receive性能估算，**步骤：**

1. 测量现有系统的链路带宽并计算利用率
2. 结合目标系统的峰值链路带宽
3. 估算目标系统上集合通信算子的延迟

NeuSight Workflow

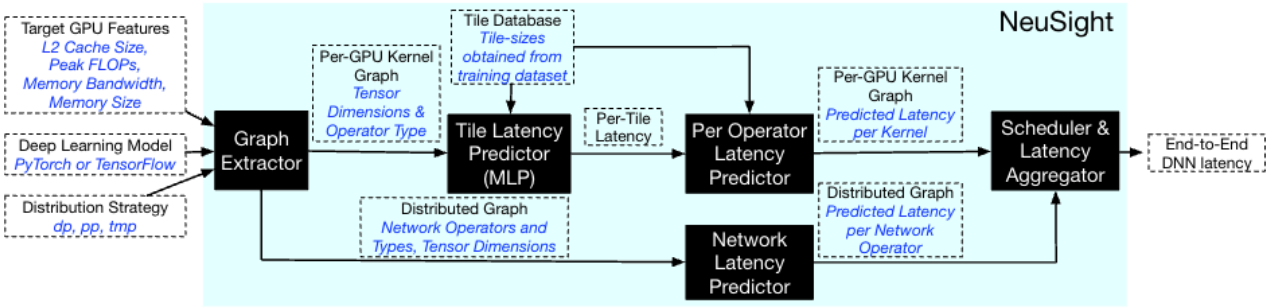


Figure 6. Overall workflow of NeuSight. The Tile Latency Predictor uses an MLP to determine the utilization and predicts the kernel latency. This prediction is aggregated for per-device and distributed execution latency forecasting.

接收 PyTorch 中目标深度学习模型的描述，并提取算子 / 内核图。记录每个内核的元数据，包括算子类型和输入 / 输出张量维度。基于这些元数据和分块大小，每个算子通过内核级预测器生成的预测结果进行标注。单 GPU 延迟通过累加每个内核在单个设备上的预测延迟计算得出。

Evaluation

Experiment Setup

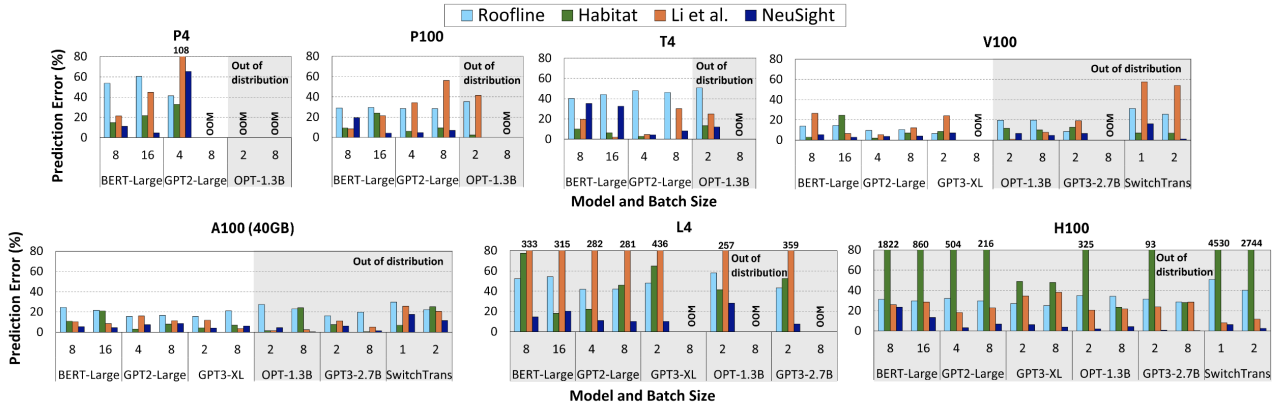
Table 4. List of GPUs used to train and test the frameworks. GPUs reserved for the test set are highlighted in grey.

Vendor	GPU	Year	Peak FLOPS (TFLOPS)	Memory Size (GB)	Memory BW (GB/s)	# SMs	L2 Cache (MB)
NVIDIA	P4	2016	5.4	8	192	40	2
	P100	2016	9.5	16	732	56	4
	V100	2017	8.1	32	900	80	6
	T4	2018	14.1	16	320	40	4
	A100-40GB	2020	19.5	40	1555	108	40
	A100-80GB	2020	19.5	80	1935	108	40
	L4	2023	31.3	24	300	60	48
	H100	2022	66.9	80	3430	132	50
AMD	MI100	2020	23.1, 46.1(Matrix)	32	1230	120	8
	MI210	2021	22.6, 45.3(Matrix)	64	1640	104	16
	MI250 (per die)	2021	22.6, 45.3(Matrix)	64	1640	104	16

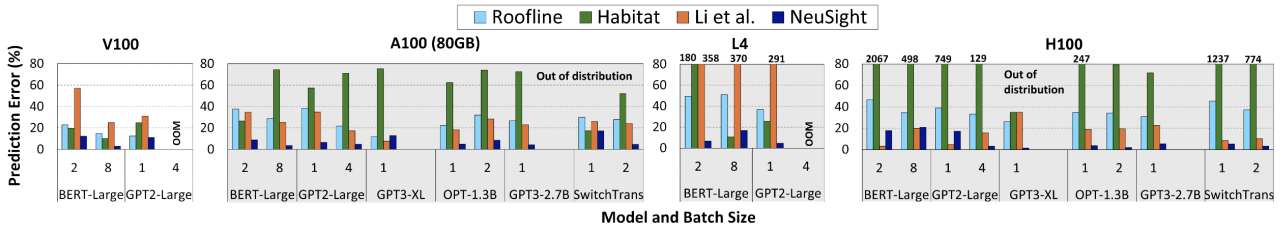
Table 5. Workloads evaluated. Table details model complexity through parameter size and architecture configurations.

Model	Year	Parameter Size	# of Layers	# Attention Heads	Hidden Dimensions	Sequence Length
BERT Large	2018	340M	12	16	760	512
GPT2 Large	2019	774M	36	20	1280	1024
GPT3 XL	2020	1.3B	24	24	3072	2048
OPT 1.3B	2022	1.3B	24	24	2048	2048
GPT3 2.7B	2020	2.7B	32	32	2560	2048
SwitchTrans	2021	5.3B	24	32	1024	512

Single-Device Execution Results



(a) Inference latency prediction error



(b) Training latency prediction error

Figure 7. The inference and training latency prediction percentage error of NeuSight and baselines with different batch sizes. Out of distribution GPUs and models are highlighted. Certain models resulting in OOM are omitted due to space constraints.

实验结果：

NeuSight 在分布外 GPU 上的优势尤为显著，其平均延迟预测误差仅为 8.1%，最大误差 28.2%。架构感知预测：NeuSight 未直接依赖 MLP 进行延迟预测，而是在机器学习预测基础上叠加了 GPU 架构特性与执行行为分析；新型 GPU 适应性：对于 H100 等具备训练集未覆盖特征的新一代 GPU，NeuSight 仍能保持稳定预测；鲁棒性验证：在分布外特征处理上表现出更强的鲁棒性，即使面对未见过的 GPU 架构也能实现精准预测。

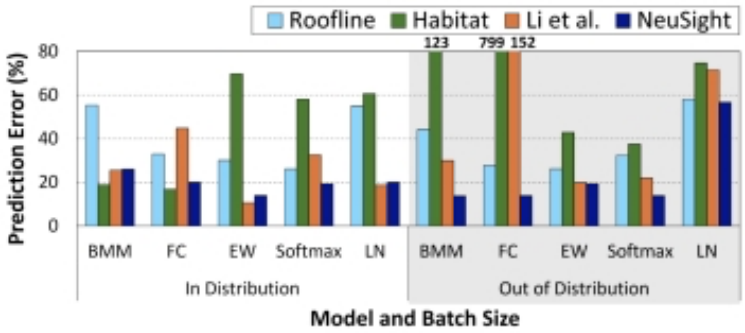


Figure 8. Prediction percentage error for each operator type (BMM, Fully-Connected (FC), Element-Wise (EW), Softmax, Layer Normalization (LN)), averaged over all the evaluated workloads.

Table 6. Per-operator contribution to the overall latency for various models on H100 for model inference. The contributions from different types of operators are shown as percentages.

Model	Batch Size	BMM	LINEAR	EW	SOFTMAX	LN	OTHERS
BERT-Large	16	12%	74%	10%	3%	2%	0%
GPT2-Large	4	11.8%	62.8%	14.9%	4.0%	1.1%	5.3%
OPT13	2	13.1%	62.9%	9.6%	5.9%	0.6%	7.8%
GPT3-XL	2	9.5%	76.4%	7.9%	2.5%	0.4%	3.3%

实验结果：

层归一化的误差率较高，这是因为其延迟时间通常较短，导致精确预测具有挑战性。由于层归一化对整体延迟的贡献至多为 2%，其较高的误差不会显著影响 NeuSight 的端到端预测精度。